Quadrilateral Mesh Smoothing

Xuan Huang

Advised by Professor Dianna Xu

April 4, 2018

Contents

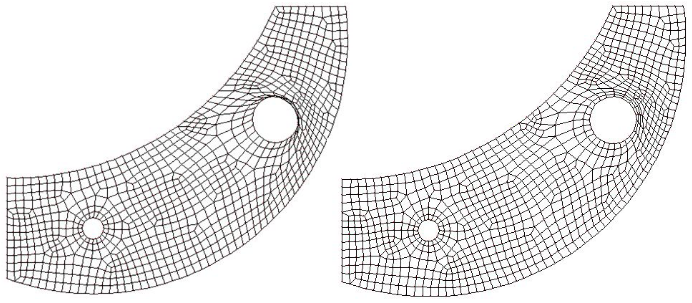


Figure 1: Quadrilateral meshes smoothing via vertex repositioning

1 Introduction

Mesh quality improvement is an important problem with a wide range of practical applications. Element quality of a mesh heavily affects the results of numerical simulation, such as rendering (Figure 2) or physical simulation (Figure 3) using that mesh. In the context of finite element mesh smoothing, vertex repositioning is the primary technique employed, where we allow vertex motion only and the connectivity of all edges remains unchanged. In the 2D cases we are interested in, this means that only the x y values of vertices can be modified.

Based on the ideas of Laplacian Smoothing and our previous work on triangular mesh smoothing, this paper presents an easy-to-implement, computationally inexpensive method to provide quality improvements on any given quadrilateral mesh.

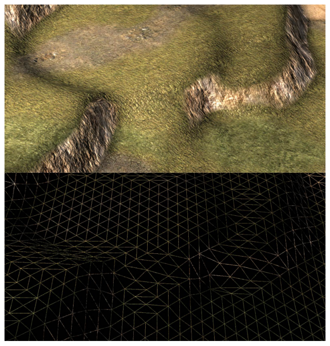


Figure 2: A example of terrain rendering

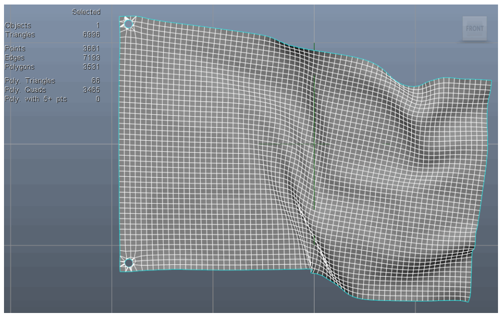


Figure 3: High-quality cloth simulation with NVIDIA

2 Element Quality

Element quality is measured either by max/min angles or aspect ratio (longest edge over shortest), or both. In a mesh the aspect ratio is defined as the largest ratio of longest edge over shortest edge among all elements, and max/min angles are the maximum/minimum among all angles. These two proprieties are related in triangles (by the law of sine), but the relationship is quite loose in quads (Figure 4). We investigate a smoothing method based on the idea of improving aspect ratio via circumscircles.

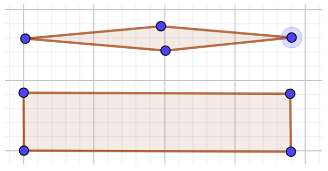


Figure 4: Aspect ratio and angles are not strictly related

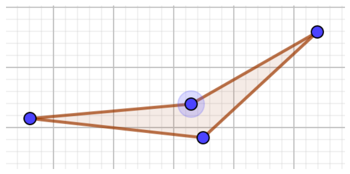


Figure 5: Non-convex quad

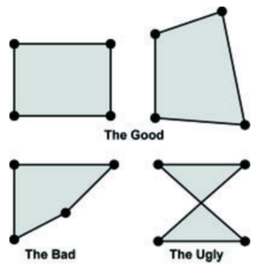


Figure 6: The good, the bad, and the ugly

Convexity is another important aspect of quadrilateral elements (Figure 5). It is very undesirable to generate any element that has an angel greater than 180 degree. Again this is never a problem for triangles, and one of our goals will be trying to eliminate such cases from happening. In addition, self-intersection and edge flipping should be avoided under any circumstances. Figure 6 illustrates a general situation for desired and undesired quadrilateral elements.

3 Background

Existing methods usually divide into mesh modification via vertex insertion/deletion, edge/face swapping and remeshing [Dey and Ray 2010], or vertex repositioning without changing mesh connectivity [Amenta et al. 1997; Field 1988; Zhou and Shimada 2000]. Among those that do not modify mesh topology, Laplacian smoothing is the most commonly used because of its simplicity. In its most basic form, it moves each vertex to the centroid of the polygonal region formed by its neighboring vertices. It is a local method with very low computational cost, compared to the alternatives, which are optimization-based [Chen and Holst 2011; Freitag 1997; Parthasarathy and Kodiyalam 1991]. The most closely related work is Zhou and Shimadas angle-based Laplacian smoothing [Zhou and Shimada 2000], which is a variant of Laplacian smoothing.

Other methods including constraining vertices with parametric spaces derived from individual mesh elements (faces, edges) and optimizing an objective function with respect to the parametric coordinates [Rao and Mikhail], cleaning-up nodes with higher number of connectives. In the meantime, there is no universal consensus on the measurement of quadrilateral elements. A list of metrics encoding information of the element size, shape, and skew are used to analyze the quality of quadrilateral elements [Patrick M. Knupp].

4 Triangular Mesh Smoothing

4.1 Laplacian and its Angle Based Variation

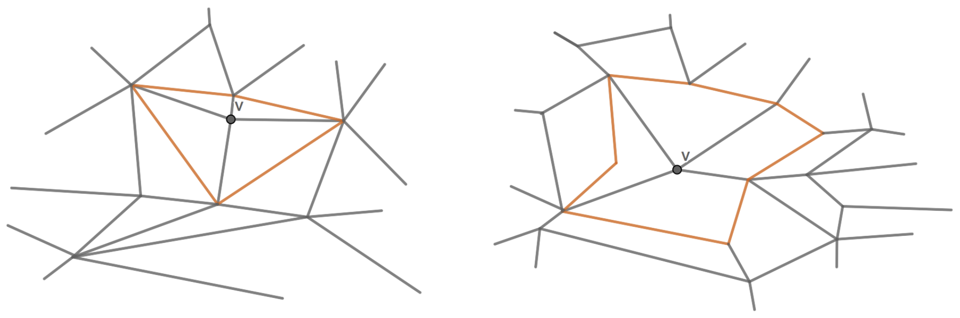


Figure 7: The **star region** of v is the polygonal region that includes all its adjacent faces

The widely used Laplacian smoothing method is a computationally inexpensive, easy to implement local technique for triangular mesh smoothing. It simply moves each vertex to the centroid specified by the polygon formed by all it adjacent vertices, which is also known as the **star region**. Figure 7 shows the star region of a vertex v in an triangular mesh and a quadrilateral mesh. The movement is applied on every star region of each vertex for several iterations, usually until the positions of all vertices change little when further smoothed, or the mesh quality no longer improves.

Zhou and Shimada presents an angle-based smoothing method based on the Laplacian smoothing algorithm [Amenta et al. 1997; Field 1988; Zhou and Shimada 2000]. Instead of taking the centroid to be the new position of each vertex, they rotate each interior edge attached to the angle bisector, and take the average value resulted from all such repositioning actions to be the new position of the vertex.

Figure 8 shows the Laplacian method and its variation with a specific example. In the star region of the vertex to be smoothed, the Laplacian method calculated the centroid of the star region and assign the vertex position there. The angle-based variation rotated the interior edges to find the new position, and take the average in the end.

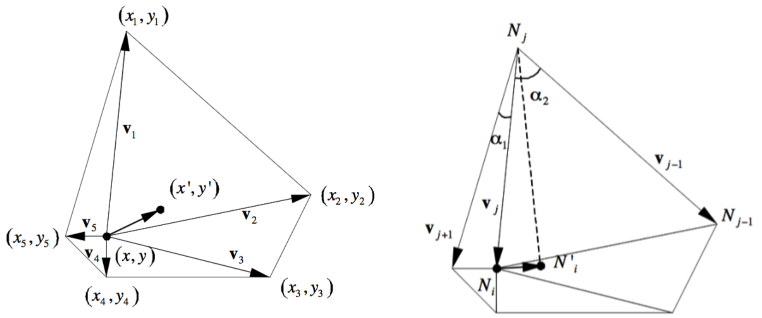


Figure 8: **Left**: The Laplacian method; **Right**: The angle-based variation. Rotating edge N_jN_i to angle bisectors to get the new position N'_i for vectex N_i

Problem arises for this method as it does not always move the vertices to optimal positions, and due to the fact that the centroid of a polygon could lie very close to, or even outside the shape itself, sometimes it generate invalid elements.

The result has a general improvements in mesh quality improving comparing to the original Laplacian method, and has largely reduced the possibility of getting invalid elements. It also provides a mechanism to implement the method on quad meshes simply by treating the quadrilateral element as the combination of two triangles.

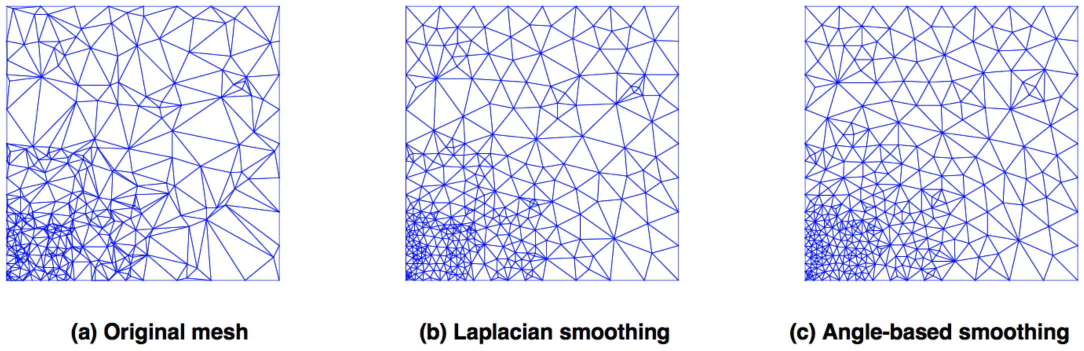


Figure 9: The Laplacian method applied on a triangular mesh

4.2 Aspect-Ratio Based

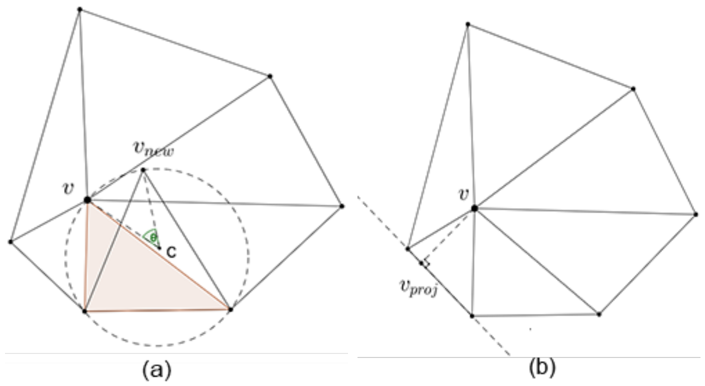


Figure 10: (a) Move v to v_{new}; (b) $\overline{vv_{proj}}$ is the shortest distance to star boundary

This section introduce our previous method to improve aspect ratio as an alternative to the existing angle-based method. We believe that aspect ratio will lead to a more balanced improvement in triangles, and this is the work motivated by aspect ratio improvements for quadrilateral meshes, which are not directly related to angles.

The idea is that inside the star region of each vertex, move the center vertex along the circumcircle of each incident triangle in a direction that will decrease the difference between each pair of adjacent interior edges. Each move must improve the aspect ratio of a triangle, because one of the two edges must be the shortest or longest edge of the triangle considered. Some restrictions are applied so that triangles already with good quality are not further modified and potentially sacrificed for worse ones.

The algorithm performs the following steps:

1. As shown in Figure 10, for each vertex v not on the boundary there are k faces inside its star polygon. For each triangle if the two interior edges has a ratio higher than 1.5, calculate its circumcircle centered on C.
2. Move v clockwise or counter-clockwise towards the longer incident edge.
3. Let $\overline{vv_{proj}}$ be v's vertical distance to each star polygon edge. If the new position ends up decreasing the shortest distance from the vertex to the star polygon boundary, discard movement.
4. Take the mean of all positions after processing all k faces and assign the v there. Iterate until the resulting positions converges.

The max aspect ratio is improved over angle-based method in all cases tested. The algorithm might also provide some additional improvements on angles. It tends to recover seriously distorted areas and is also much less likely to result in invalid shapes compared to the angled- based method.

4.3 Results

It's the beginning of a section. What is "the" algorithm? You never introduced it before!
Generally avoid using "the" when first introducing a concept. It's pretty confusing

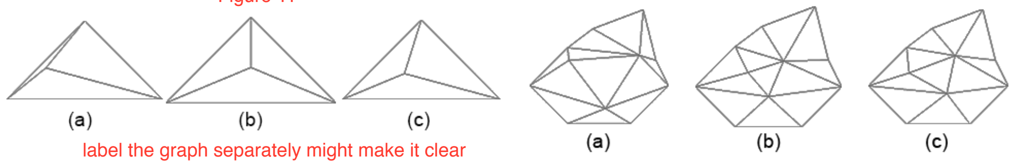
The algorithm typically generate a mesh with aspect ratio under 3 as well as some improvements on angle. It tends to recover seriously distorted areas and very unlikely to result in invalid shape. Followed are 4 examples, column (a) represents the original mesh, (b) represents the result from Angle-based smoothing, and (c) represents our result from Aspect-ration based smoothing, followed by a static comparison table :

missing verb (add "is")

for example:

Figure 11

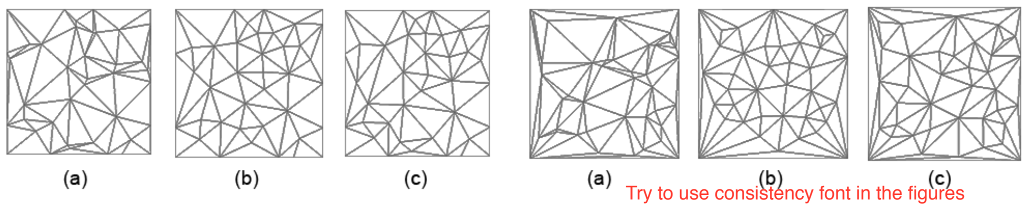
Figure 12



label the graph separately might make it clear

Figure 11: simple triangular meshes: 3 triangles and 10 triangles

Would be better to describe and provide a comparison

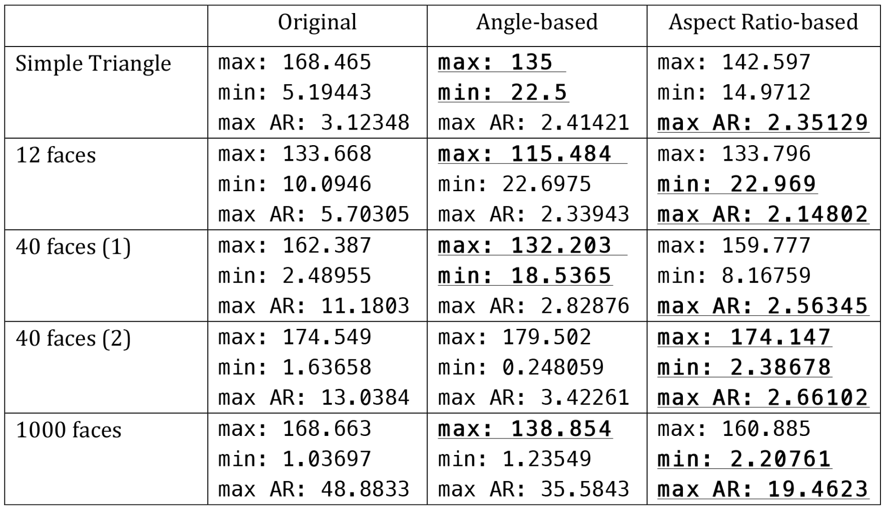


Try to use consistency font in the figures

Figure 12: 40 triangles. A nice input example and a bad one.

same as above. Mixing these two comparisons together is pretty confusing. I understand what you mean but it looks messy and not obvious at first read

Recommend looking over some literatures and see how they label the graphs. Usually it includes a title, and a description.



	Original	Angle-based	Aspect Ratio-based
Simple Triangle	max: 168.465 min: 5.19443 max AR: 3.12348	**max: 135** **min: 22.5** max AR: 2.41421	max: 142.597 min: 14.9712 **max AR: 2.35129**
12 faces	max: 133.668 min: 10.0946 max AR: 5.70305	**max: 115.484** min: 22.6975 max AR: 2.33943	max: 133.796 **min: 22.969** **max AR: 2.14802**
40 faces (1)	max: 162.387 min: 2.48955 max AR: 11.1803	**max: 132.203** **min: 18.5365** max AR: 2.82876	max: 159.777 min: 8.16759 **max AR: 2.56345**
40 faces (2)	max: 174.549 min: 1.63658 max AR: 13.0384	max: 179.502 min: 0.248059 max AR: 3.42261	**max: 174.147** **min: 2.38678** **max AR: 2.66102**
1000 faces	max: 168.663 min: 1.03697 max AR: 48.8833	**max: 138.854** min: 1.23549 max AR: 35.5843	max: 160.885 **min: 2.20761** **max AR: 19.4623**

You need to explain this chart. Especially given that you use some synonyms like AR for Aspect Ratio I believe

Figure 13: comparison table, best results underlined

As shown the aspect ratio tend to beat angle-based method at most cases with also a slight improvement on angles. Under extreme cases the aspect ratio method also protects small angles.

Go over the examples, and talk about them in detail. Maybe you want to compare the results of your algo with the original ones.

5 Quadrilateral Mesh Smoothing

5.1 Previous work and its limitation on Quadrilateral meshes

The triangular mesh smoothing methods do provide techniques via cutting the quads into two triangles and smooth the resulted triangular mesh. Apparently this way of smoothing ignores the proprieties special to quads, such as the loose angle-aspect ratio relationship, and the fact that a quad could become non-convex during the process. We would like our algorithm to be aware of such proprieties, and provide a more reliable solution to quadrilateral meshes specifically.

Our algorithm design focuses on the worse angles and aspect-ratio in the mesh, since they are usually the bottleneck in terms of further simulation work. We propose a method to improve both angle and aspect-ratio locally in either each quad, and then avoid the problem of self-intersecting and non-convexity by a distance check mechanism within each star region. The procedure is repeated until there is no further improvements.

5.2 Our method

After sorting all incoming quads by their maximum angles, we smooth the mesh starting from the quad with worst maximum angle. The algorithm performs the following steps:

1. As shown in Figure 14, for each vertex v not on the boundary we look at its angle within the quad, push the v further in the direction perpendicular to the diagonal if the angle is larger than a certain *maximum angle*, and pull v closer in the direction perpendicular to the diagonal $v'v''$ if the angle is smaller than a certain *minimum angle*.

2. For the two adjacent vertices of v not on the boundary, shrink (if large angle) or elongate (if small angle) the diagonal $v'v''$ accordingly.

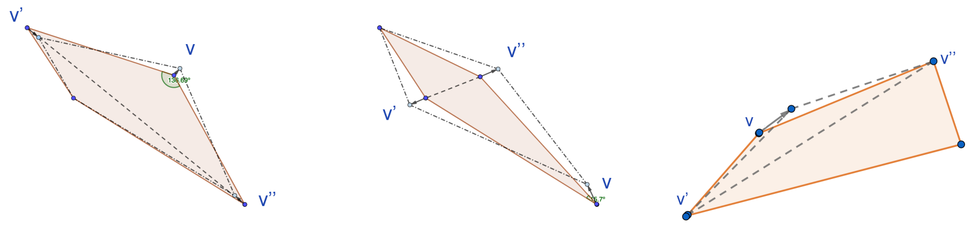


Figure 14: Smoothing scheme: Large angle, Small angle, and AR improvement

3. Move v parallel to the diagonal $v'v''$ towards the longer incident edge.

4. Let $\overline{vv_{proj}}$ be v's vertical distance to each star polygon edge as well as to each quad's diagonal that doesn't include v. If the new position ends up decreasing the shortest distance, discard movement.

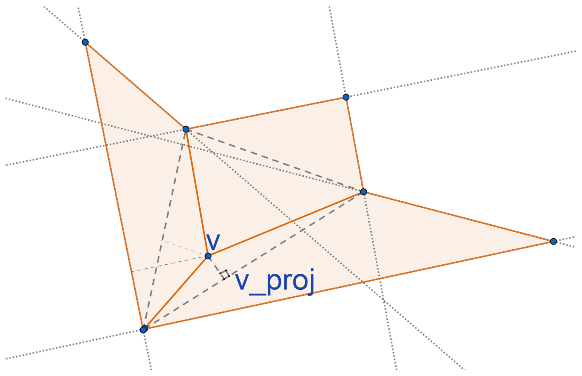


Figure 15: Boundary distance check in step 4

5. Take the mean of all positions after processing all quads

6. Iterate until the resulting positions converges.

Step 1-3 provide intuitive way to improve the two factors of individual quad element. The shortest distance check in step 4 serves as a protection against non-convexity and self-intersection, since the vertex is not allowed to get too close to any diagonal or quad boundary.

5.3 Result and Further Plan

We apply the method on single quads as well as some complicated quadrilateral meshes. There is a general improvements on both the max/min angle and the aspect ratio, while no non-convex or self-intersecting elements are found up to this point. The algorithm shows significant improvements in eliminating extremely bad shapes in particular.

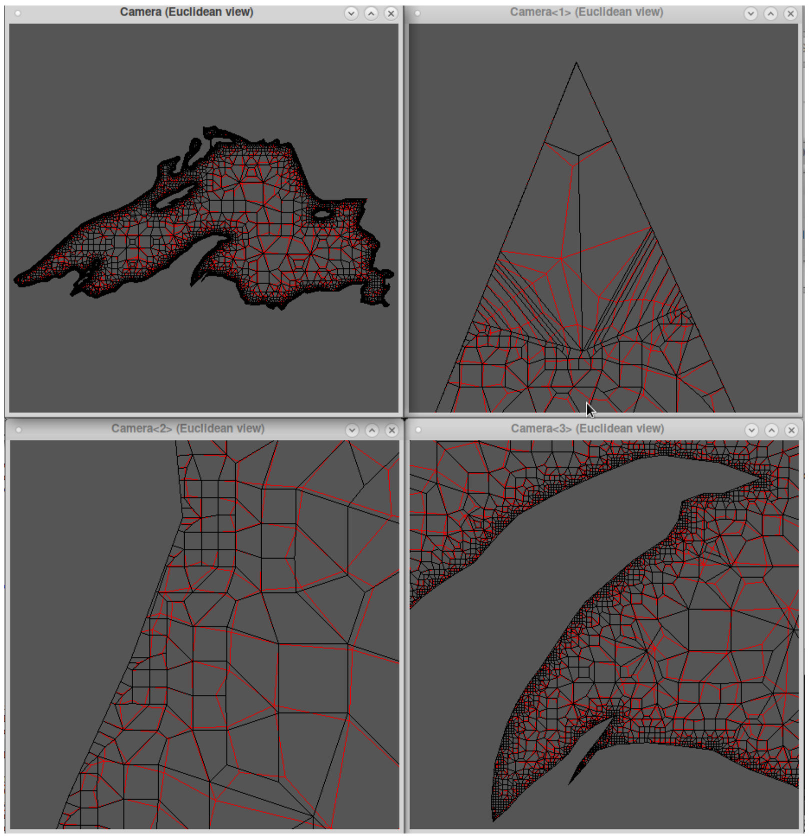

Camera (Euclidean view)
Camera<1> (Euclidean view)
Camera<2> (Euclidean view)
Camera<3> (Euclidean view)

Figure 16: 24K quads example - lake superior (Black-input Red-result)

We also looked into a variation of the method by going through the mesh by each star region instead of each quad. This version converges with general weaker improvements, but the elements looks better aligned and it finishes within about half the time than the one described above. Below are examples of the method as well as its variation applied, with the comparison to Laplacian method.

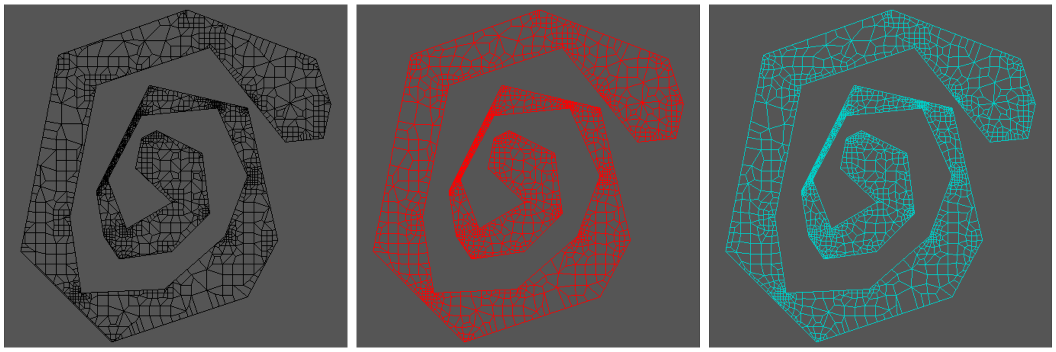


Figure 17: 1.2K quads example-spiral (**Input** **Smooth by quad** **Smooth by star region**)

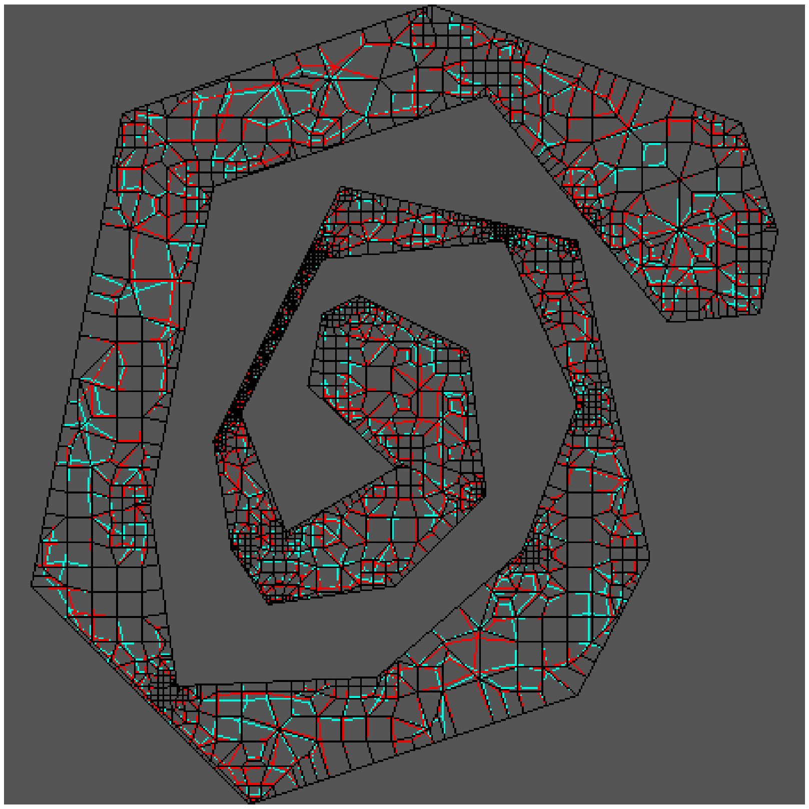


Figure 18: spiral: mesh comparison

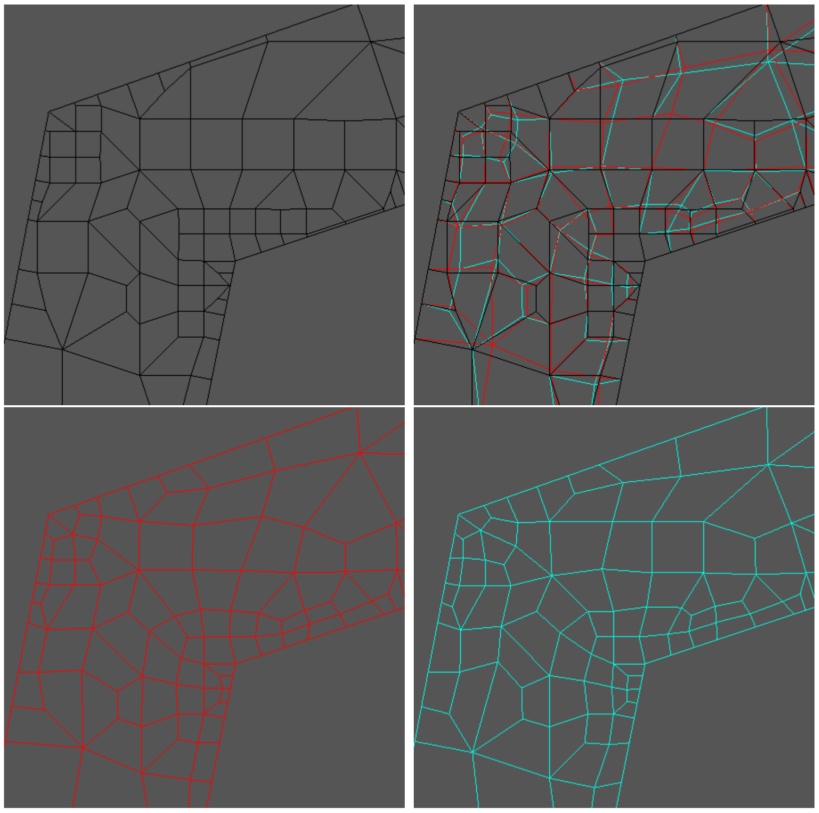


Figure 19: spiral: boundary region

We can see that both method improves the mesh quality, and tend to recover the bad shapes along boundary. Specifically, **Smooth by star region** seems to provide a more uniformed set of elements (Figure 19), while **Smooth by quad** seems to guarantee a recover for extreme angles (Figure 20). Figure 20 also show the results of smoothing on narrow and nice regions.

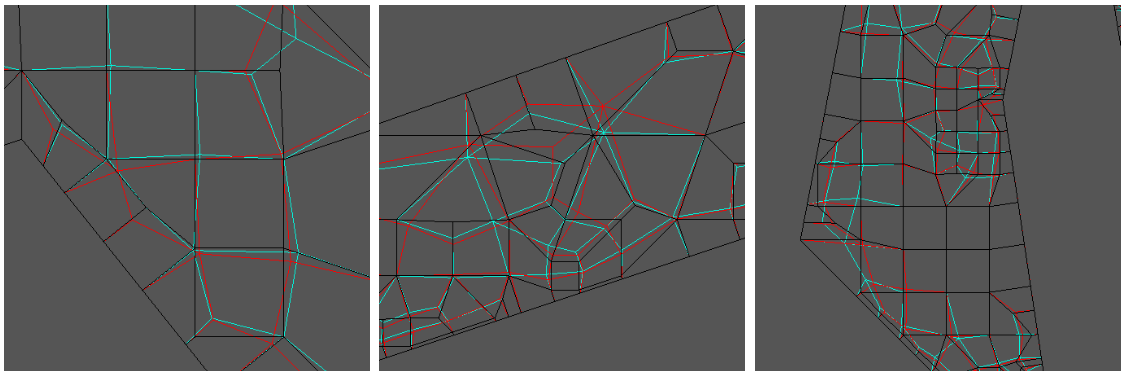


Figure 20: spiral: extreme angles; narrow region; nice region

It might be obvious to you given the figure. But for readers, it's a little confusing what exactly is improving

Note that the **Laplacian** method fails to prevent non-convexity.

Assume less of your reader and write verse for each figure and explain what you did and what improved

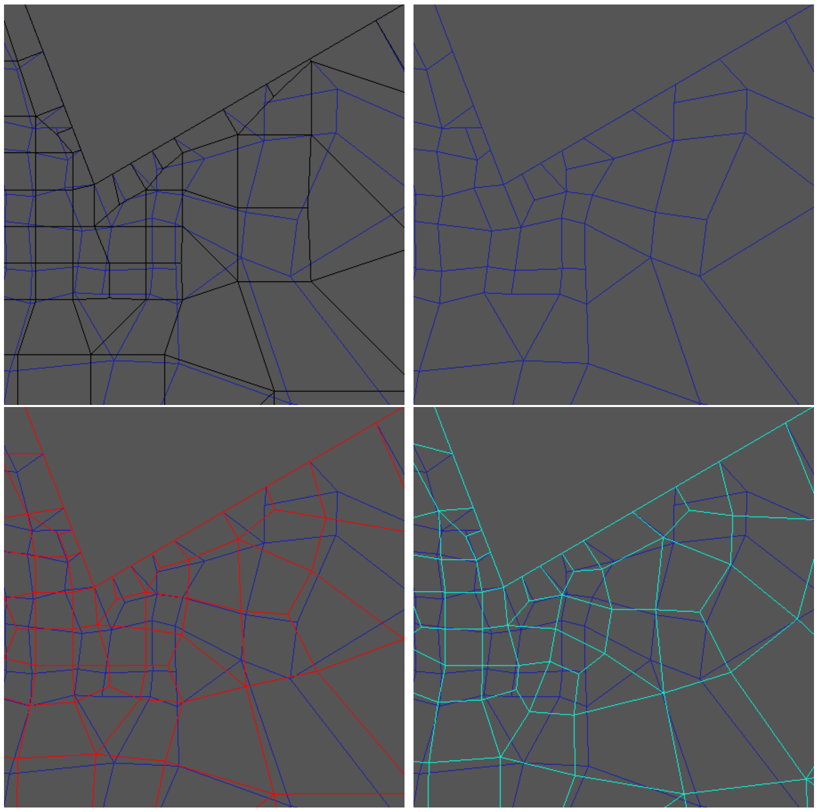
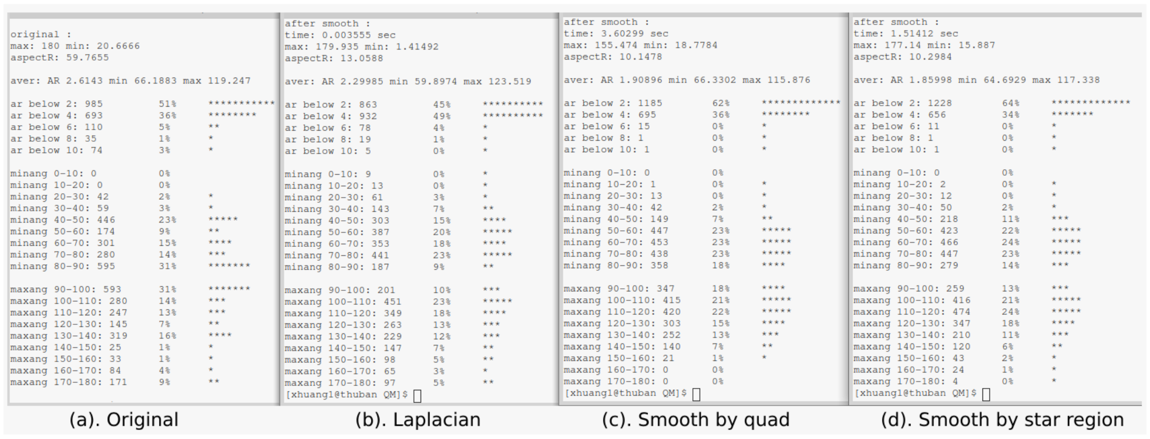


Figure 21: spiral: Laplacian and our method compared

This looks like screenshots. Use proper figure or try to format it more professionally averages of what?

Comparison chart (this one just screen-shot) table for: time / angles / ar / averages

Better write full sentence not using “ar” and “/“



```
original :
max: 180 min: 20.6666
aspectR: 59.7655

aver: AR 2.6143 min 66.1883 max 119.247

ar below 2: 985         51%      ***********
ar below 4: 693         36%      ********
ar below 6: 110         5%       **
ar below 8: 35          1%       *
ar below 10: 74         3%       *

minang 0-10: 0          0%
minang 10-20: 0         0%
minang 20-30: 42        2%       *
minang 30-40: 59        3%       *
minang 40-50: 446       23%      *****
minang 50-60: 174       9%       **
minang 60-70: 301       15%      ****
minang 70-80: 280       14%      ***
minang 80-90: 595       31%      *******

maxang 90-100: 593      31%      *******
maxang 100-110: 280     14%      ***
maxang 110-120: 247     13%      ***
maxang 120-130: 145     7%       **
maxang 130-140: 319     16%      ****
maxang 140-150: 25      1%       *
maxang 150-160: 33      1%       *
maxang 160-170: 84      4%       *
maxang 170-180: 171     9%       **
```

(a). Original

```
after smooth :
time: 0.003555 sec
max: 179.935 min: 1.41492
aspectR: 13.0588

aver: AR 2.29985 min 59.8974 max 123.519

ar below 2: 863         45%      **********
ar below 4: 932         49%      **********
ar below 6: 78          4%       *
ar below 8: 19          1%       *
ar below 10: 5          0%       *

minang 0-10: 9          0%       *
minang 10-20: 13        0%       *
minang 20-30: 61        3%       *
minang 30-40: 143       7%       **
minang 40-50: 303       15%      ****
minang 50-60: 387       20%      *****
minang 60-70: 353       18%      ****
minang 70-80: 441       23%      *****
minang 80-90: 187       9%       **

maxang 90-100: 201      10%      ***
maxang 100-110: 451     23%      *****
maxang 110-120: 349     18%      ****
maxang 120-130: 263     13%      ***
maxang 130-140: 229     12%      ***
maxang 140-150: 147     7%       **
maxang 150-160: 98      5%       **
maxang 160-170: 65      3%       *
maxang 170-180: 97      5%       **
[xhuang1@thuban QM]$ 
```

(b). Laplacian

```
after smooth :
time: 3.60299 sec
max: 155.474 min: 18.7784
aspectR: 10.1478

aver: AR 1.90896 min 66.3302 max 115.876

ar below 2: 1185        62%      *************
ar below 4: 695         36%      ********
ar below 6: 15          0%       *
ar below 8: 1           0%       *
ar below 10: 1          0%       *

minang 0-10: 0          0%
minang 10-20: 1         0%       *
minang 20-30: 13        0%       *
minang 30-40: 42        2%       *
minang 40-50: 149       7%       **
minang 50-60: 447       23%      *****
minang 60-70: 453       23%      *****
minang 70-80: 438       23%      *****
minang 80-90: 358       18%      ****

maxang 90-100: 347      18%      ****
maxang 100-110: 415     21%      *****
maxang 110-120: 420     22%      *****
maxang 120-130: 303     15%      ****
maxang 130-140: 252     13%      ***
maxang 140-150: 140     7%       **
maxang 150-160: 21      1%       *
maxang 160-170: 0       0%
maxang 170-180: 0       0%
[xhuang1@thuban QM]$ 
```

(c). Smooth by quad

```
after smooth :
time: 1.51412 sec
max: 177.14 min: 15.887
aspectR: 10.2984

aver: AR 1.85998 min 64.6929 max 117.338

ar below 2: 1228        64%      *************
ar below 4: 656         34%      *******
ar below 6: 11          0%       *
ar below 8: 1           0%       *
ar below 10: 1          0%       *

minang 0-10: 0          0%
minang 10-20: 2         0%       *
minang 20-30: 12        0%       *
minang 30-40: 50        2%       *
minang 40-50: 218       11%      ***
minang 50-60: 423       22%      *****
minang 60-70: 466       24%      *****
minang 70-80: 447       23%      *****
minang 80-90: 279       14%      ***

maxang 90-100: 259      13%      ***
maxang 100-110: 416     21%      *****
maxang 110-120: 474     24%      *****
maxang 120-130: 347     18%      ****
maxang 130-140: 210     11%      ***
maxang 140-150: 120     6%       **
maxang 150-160: 43      2%       *
maxang 160-170: 24      1%       *
maxang 170-180: 4       0%       *
[xhuang1@thuban QM]$ 
```

(d). Smooth by star region

Figure 22: spiral.off comparison table- smooth for 50 iterations

Try explain all the concepts involved that readers need to understand, and highlight the result and improvement. Right now it’s not very clear what’s changed and you want readers to focus on

Why and how? You need to discuss it

This result indicates that our algorithm provides a computationally efficient way to smooth quadrilateral meshes while it is very unlikely to produce invalid shapes. Though the Laplacian has a great advantage in terms of running time, our smoothing process converges in a reasonable time, with an overall improvement to the mesh elements as well as to the extreme cases.

We can also pick a subset of meshes to smooth for a certain number of iterations so that the worst group of elements will be taken care of first. The result from this procedure depends largely on the input mesh, and thus usually requires manual coefficients setting within the code.

(If we allow more dramatic movements within some bad regions, the result might be improved once more. part to be done)

6 Documentation

Shouldn't it be called References?

L. Chen and M. Holst. 2011. Efficient Mesh Optimization Schemes based on Optimal Delaunay Triangulations. Computer Methods in Applied Mechanics and Engineering 200 (2011), 967984.
T. Dey and T. Ray. 2010. Polygonal Surface Remeshing with Delaunay Refinement. Engineering with Computers 26 (2010), 298301.
D. Field. 1988. Laplacian Smoothing and Delaunay Triangulations. Communications in Applied Numerical Methods 4 (1988), 709712. Issue 6.
L. Freitag. 1997. On Combining Laplacian and Optimization-based Mesh Smoothing Techniques. AMD Trends in Unstructured Mesh Generation 220 (1997), 3743.
V. Parthasarathy and S. Kodiyalam. 1991. A Constrained Optimization Approach to Finite Elemement Mesh Smoothing. Finite Elements in Analysis and Design 9 (1991), 309320.
T. Zhou and K. Shimada. 2000. An Angle-based Approach to Two-dimensional Mesh Smoothing. In Proceedings, 9th International Meshing Roundtable. 373384.
J. Robinson. 1987 "CRE method of element testing and the Jacobian shape parameters", Engineering Computations, Vol. 4 Issue: 2, pp.113-118, https://doi.org/10.1108/eb023689
P M. Knupp. Algebraic mesh quality metrics for unstructured initial meshes. Parallel Computing Sciences Department, Sandia National Laboratories. 2001.

Figure 1. An example of quadrilateral mesh smoothing. Reprinted from Mesh Smoothing Challenges in the Industry, In Slide Player, N. Mukherjee, Retrieved February 17, 2018, from http://slideplayer.com/slide/5853324/. Copyright 2016 by N. Mukherjee.
Figure 2. An example of terrain rendering. Reprinted from Terrain texture projection artifacts, in Developer's blog for IceFall Games, Retrieved February 17, 2018, from http://cgicoffee.com/blog/2016/08/nvidia-flex-cloth-simulation, mtnphil. Copyright September 25, 2012 by mtnphil at Developer's blog for IceFall Games.
Figure 3. High-quality cloth simulation with NVIDIA. Reprinted from Maxwell Render/Learn/Render/Channel Rendering, in Next Limit, Retrieved February 17, 2018, from http://support.nextlimit.com/display/rf2016docs/Channel+Rendering. Copyright by RealFlow 10 Documentation.
Figure 4-5. Self made with GeoGebra at https://www.geogebra.org/.
Figure 6. The good the bad and the ugly. Reprinted from Plate/Shell in risa.com, Retrieved February 17, 2018, from https://risa.com/risahelp/risa3d/Content/3D_2D_Only_Topics/Plates.htm. Copyright by risa.com.
Figure 7 from T. Zhou and K. Shimada [2000].
Figure 8-15. Self made with GeoGebra at https://www.geogebra.org/ and Meshlab, modified with Adobe Photoshop.

You seem to have very promising results, but your writeup seems a bit raw

Several suggestions:
1. Even though you have a section for triangular and another for quadrilateral. It's not clear that these are your two results.
I would suggest add an introduction paragraph about your contribution in both triangular and quadrilateral methods
2. You want to properly insert figures
3. Disucss and explain to your reader what each figure shows. What should we focus on? What's optimized? What's the problem before and what are you improving?